agile testing

Johan Åtting (Sectra)
Johan Jonasson (House of Test)
Martin Gladh (Frontit)

Lecture at LiU 2017-10-03

Agenda

1. What is agile testing
2. What is testing really about
3. Agile testing at Sectra

Twitter: @johanatting @johanjonasson @MartinGladh

1) Agile Testing

Constraints

Short Iterations (sprints)

(Fast feedback is key)

No time to write long detailed test plans or test specifications

Common missunderstadning

Test Driven Development & Test Automation can replace manual (sapient) testing

2) What is testing really about

Video

[Open Lecture by James Bach on Software Testing]

The video is 1h 40min but let's look at this part 46:50 –> 53:42

https://www.youtube.com/watch?v=ILkT_HV9DVU

Any lessons learned?

Context

Professional scepticism

Ask questions

- **What is the context?**
- **What is happening here?**
- **What problem is this ”thing” trying to solve?**
- **I honour what you say (write) but I don’t trust you...**
- **Professional scepticism**
- **Ask questions, why, why, why**

One of many definitions:

Try it to learn sufficiently everything that matters about how the program *can* work and about how it *might not* work.

”Try the product”?

What should we try?

Can we try everything?

How do we know that it works?

When are we done?

The impossibility of complete testing

- Complete testing is impossible for several reasons:
  - We can’t test all the inputs to the program.
  - We can’t test all the combinations of inputs to the program.
  - We can’t test all the paths through the program.
  - We can’t test for all of the other potential failures, such as those caused by user interface design errors or incomplete requirements analyses.

The impossibility of complete testing

Example: If a product has 100 configuration parameters, which each have two possible values, and it takes only 1 second to perform a complete test of the whole product under a certain configuration, it will take a total of $4*10^{22}$ years to test the product under all configurations.

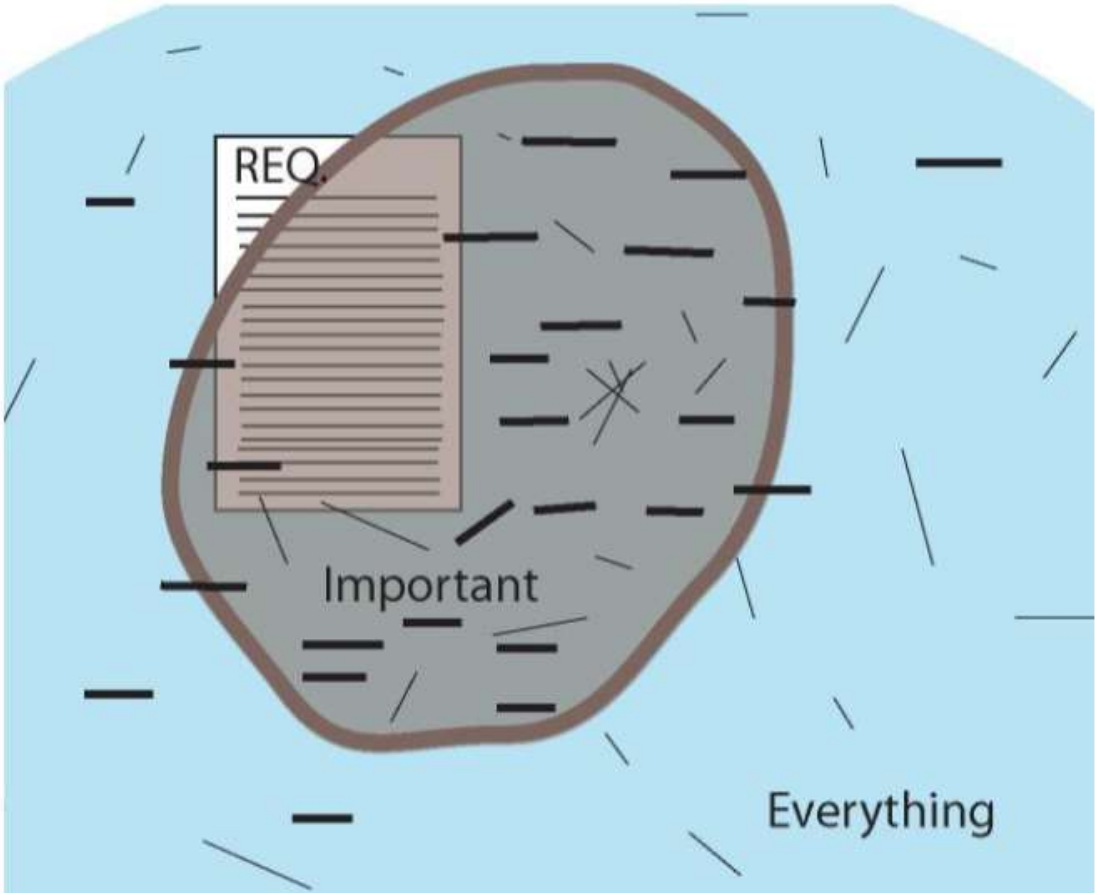
If you doubt me, read e.g. http://kaner.com/pdfs/impossible.pdf

The impossibility of complete testing

- The core problem underlying testing is that we can run only a tiny sample of the set of possible tests
- Test planning is really the development of a <u>sampling strategy</u>

- Our sampling strategy is governed by heuristics

The Software Potato



From: The Little Black Book on Test Design by Rikard Edgren

Coverage levels

Level	Label	Description	Quote
0	Not tested	Testing has not begun or is still very limited.	"We have no good info in this area"
1	Sanity Check	Major functions, simple data (often no regression tests or negative testing).	"At least it's not completely broken"
2	Common & Critical	Common and critical functions/elements exercised, also with some negative testing. Tested for major risks.	"We're gaining some confidence in this"
3	Reasonable cases	All parts exercised, including relevant negative testing. Different configurations used. Regression testing on adjacent areas. Also some focus on the most important non-functional quality characteristics.	"This feels pretty good"
4	Complex cases	More focus on evaluating different non-functional quality criteria's, such as e.g. performance, reliability, usability etc. Strong data.	"If there was a bad bug in this area, we would probably know about it"
5	Thoroughly tested	All relevant quality characteristics have been explored.	"We're confident there is no tests that could reveal anything important"

Based on the coverage levels in James Bach's low-tech testing dashboard (www.satisfice.com)

Quality

according to ISO8402:1986

The totality of features and characteristics of a product or service that bear on its **ability to satisfy stated and implied needs.**

Quality Characteristics

one of many Heuristics that can be used

- **Safety**
- **Security**
- **Usability**
- **Performance**
- **Reliability**
- **Supportability**
- **Deployementability**
- **Maintainability**
- **Documentation**

Suggested reading

Heuristic Test Strategy Model (Bach) http://www.satisfice.com/tools/satisfice-tsm-4p.pdf
Lightweight Characteristics Testing (Edgren) http://www.thetesteye.com/papers/LightweightCharacteristicsTesting.pdf

Quality Characteristics

Some examples

- **Capability (Usefullness)**
  Can the product help me with all my problem
- **Usability**
  How easy the product is to use, e.g. Learnability, Memorability, Consistency, Error handling, Documentation etc.

Quality Characteristics

- **Charisma (Desierability)**
  How compelling the product is. First impression, Look & feel, Wow, Fun to use etc.
- **Performance**
  Responsiveness, Capacity, Endurance, Scalability etc.
- **Rubustness (Reliability)**
  How well the product handles difficult situations, e.g. Stress, load, bad data, recovery etc.

Quality Characteristics

- **Deploymentability**
  How easy it is to Install, Un-install, Upgrade, Roll-back etc.
- **Supportability**
  How easy it is to support, e.g. remote login, log-files, traces, debugging, monitoring etc.

Quality Characteristics

- **Maintainability**
  How easy it is to maintain & to continue to develop (code structure, future-proof design, architecture, automatic code level unit checks, testability etc.).
- **Safety**
  (patients & users)
- **Security**
  (Confidentiality, Availability, Integrity)

Other examples

- **The Test Eye**
  http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf
  http://thetesteye.com/posters/TheTestEye_KvalitetsegenskaperForProgramvara.pdf
- **Heuristic Test Strategy Model (Bach)**
  http://www.satisfice.com/tools/satisfice-tsm-4p.pdf
- **Lightweight Characteristics Testing (Edgren)**
  http://www.thetesteye.com/papers/LightweightCharacteristicsTesting.pdf
- **ISO 2191-1 replaced by ISO 25010**
  http://en.wikipedia.org/wiki/ISO/IEC_9126

Testing is more than checking requirements

It's also about learning & exploring the product.

Checking vs Exploration

Checking is something that we do with the motivation of *confirming existing beliefs. Checking is a process of confirmation and verification.*

Exploring is something that we do with the motivation of *finding new information. Exploration is a process of discovery, investigation and learning.*

Checking

- When we already believe something to be true, we verify our belief by **checking**.
- We **check** when we've made a change to the code and we want to make sure that everything that worked before still works.
- Excellent programmers do a lot of **checking** as they write and modify their code, creating automated routines that they run frequently to check to make sure that the code hasn't broken.
- **Checking** is focused on *making sure that the program doesn't fail*.

Exploration

- We're **exploring** when we're trying to find out about the extents and limitations of the product and its design, and when we're largely driven by questions that haven't been answered or even asked before.
- **Exploration** is focused on "learning sufficiently everything that matters about how the program works and about how it *might not* work."

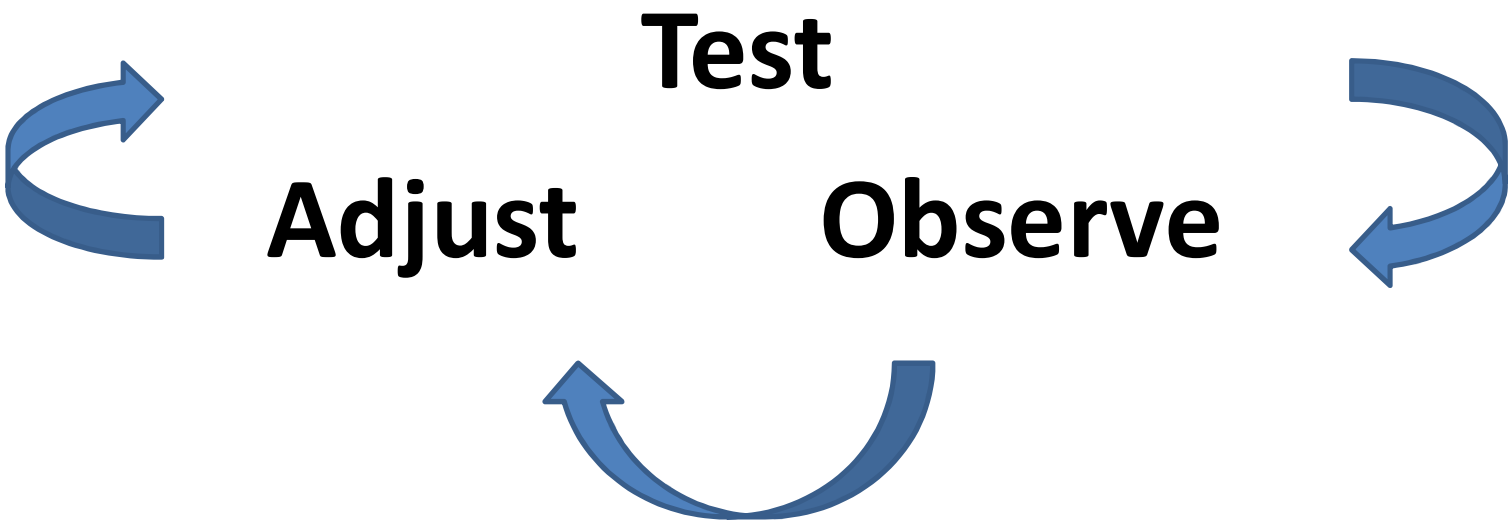
Checking vs Exploration

• If we could express our question such that a machine could ask and answer it via an assertion, it's almost certainly **checking**.

• If it *requires* a human, it's a sapient process, and is far more likely to be **exploration**.

=> Checks can be automated, exploration can not.

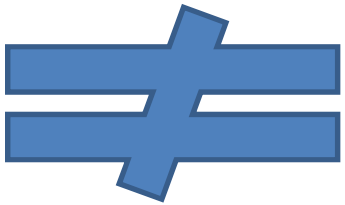
Exploratory testing



use feedback from the previous test to inform the next

Great example on exploratory approach

Exploration

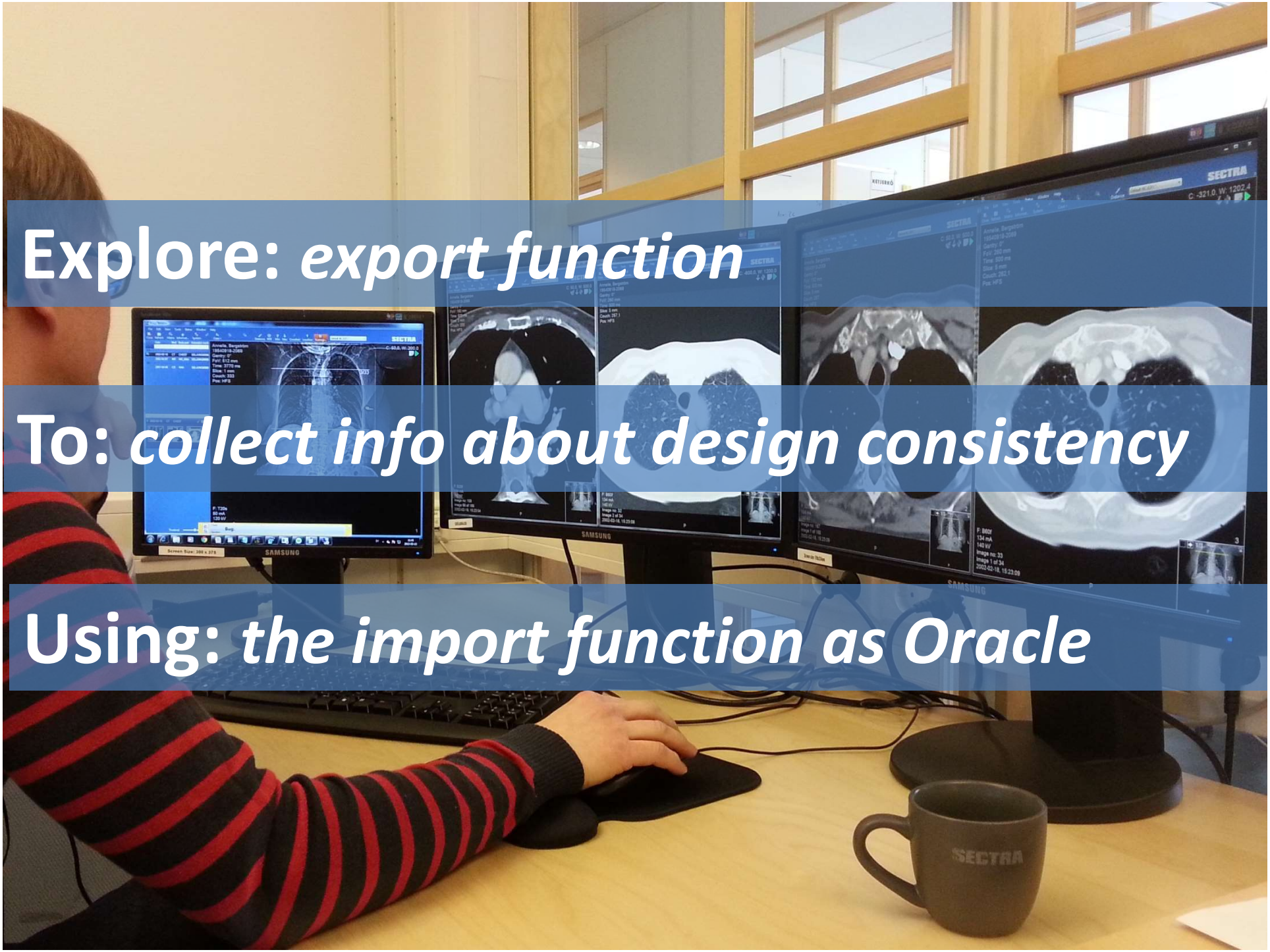


Wandering

There is always a testing mission


Explore: *The jungles of Peru*
To: *Find lost treasures*
Using: *Map, local guides, whip ...*


Explore: *export function*
To: *see how file size effects performance*
Using: *The image data db, …*


Explore: *export function*
To: *collect info about design consistency*
Using: *the import function as Oracle*

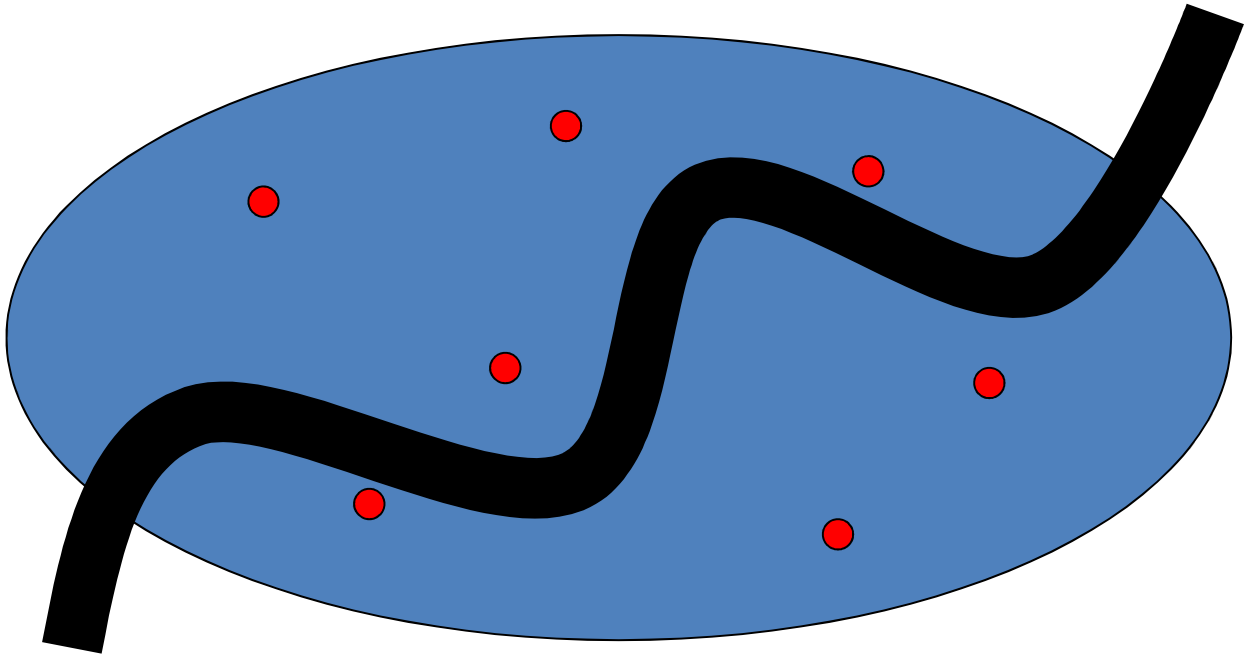
Video

Basketball players

https://www.youtube.com/watch?v=IGQmdoK_ZfY

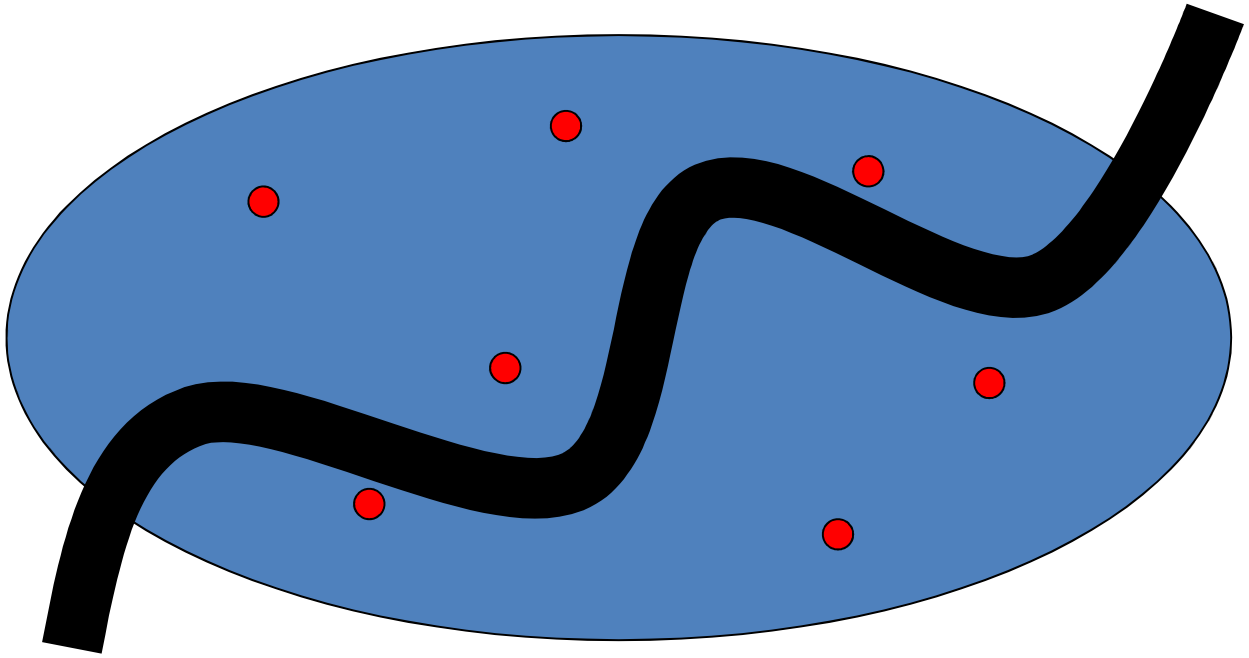
Scripted

Follow the same path every time?



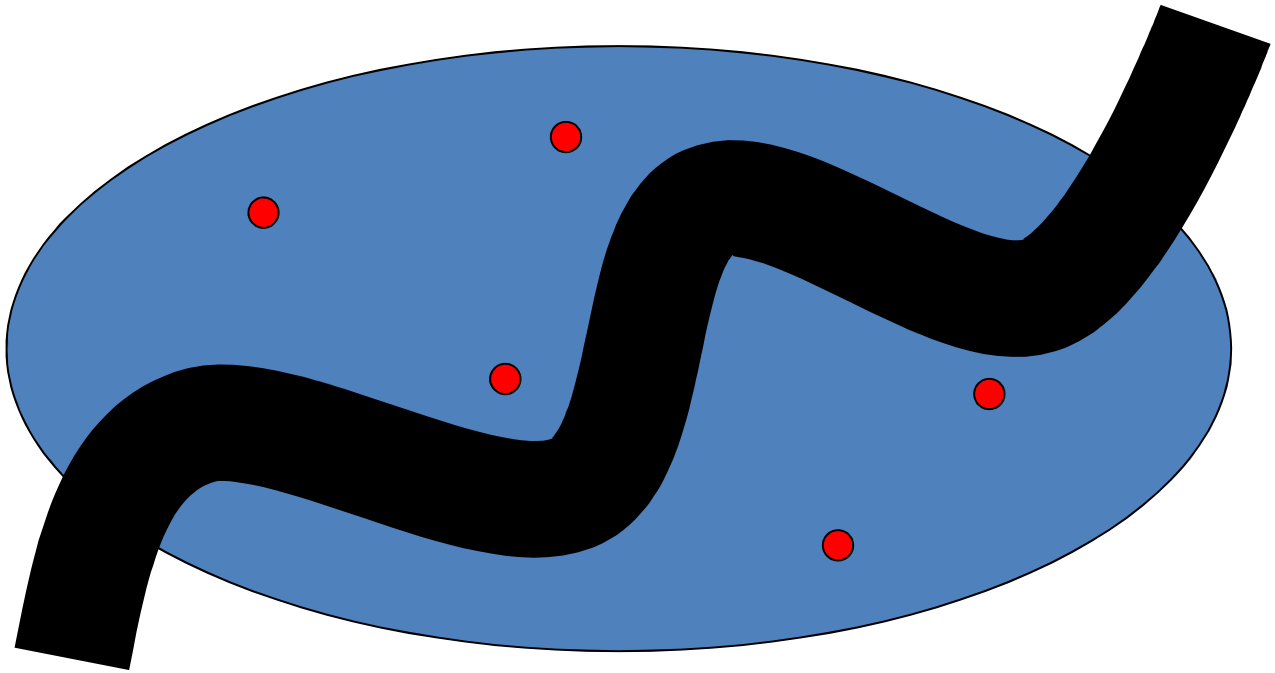
Scripted

Do not fall asleep on the testing bus!



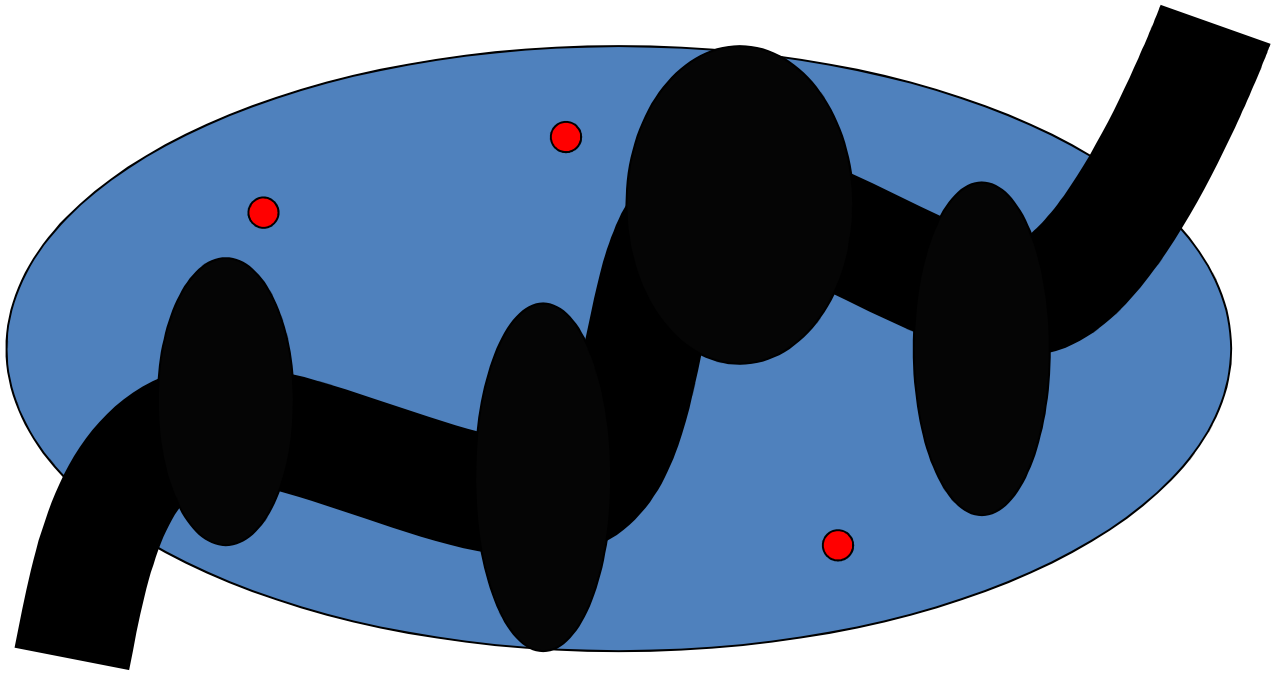
Scripted (+ some exploration)

Look out of the window



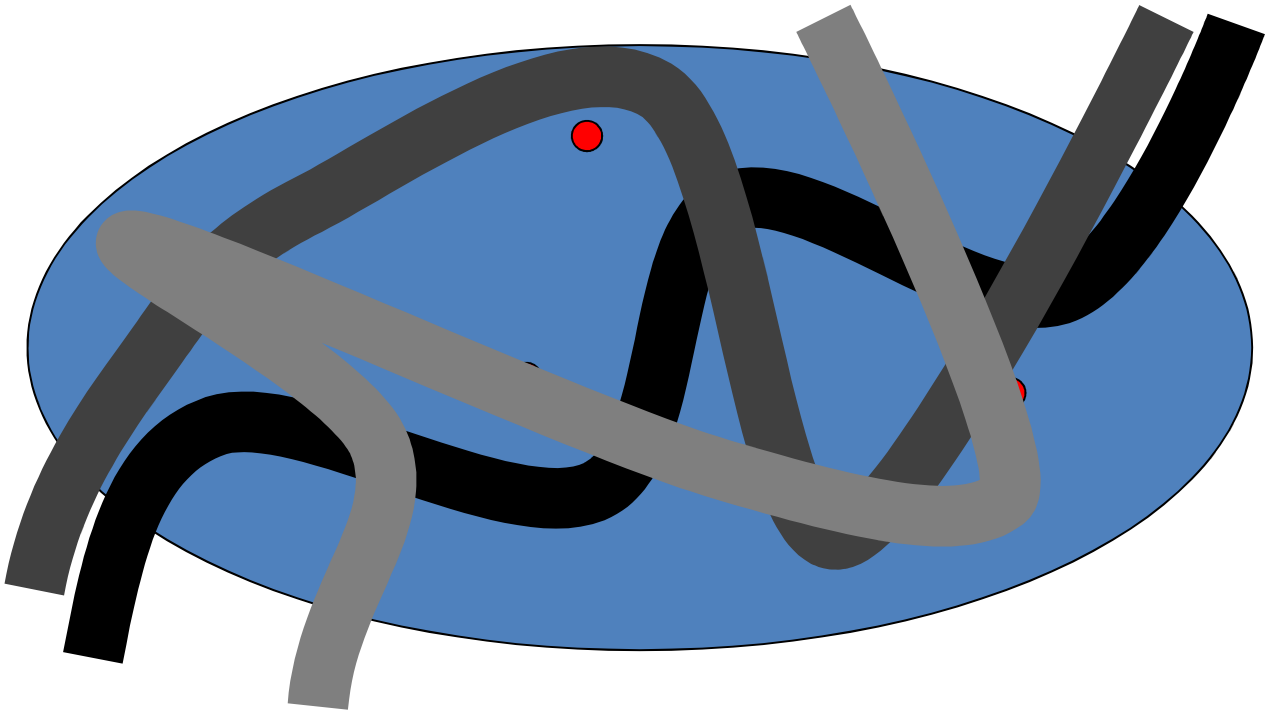
Scripted (+ some more exploration)

Get off the bus and look around…

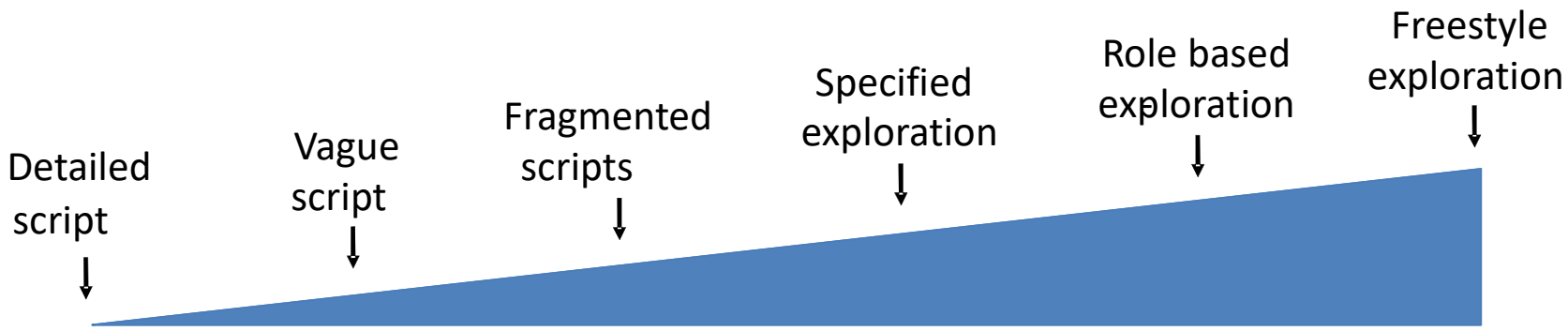


Exploratory

Or take a different route each time?



Degrees of exploration



Detailed scripts and freestyle exploration are the extreems of a broad spectrum of ”degrees of exploratory approach” to testing.

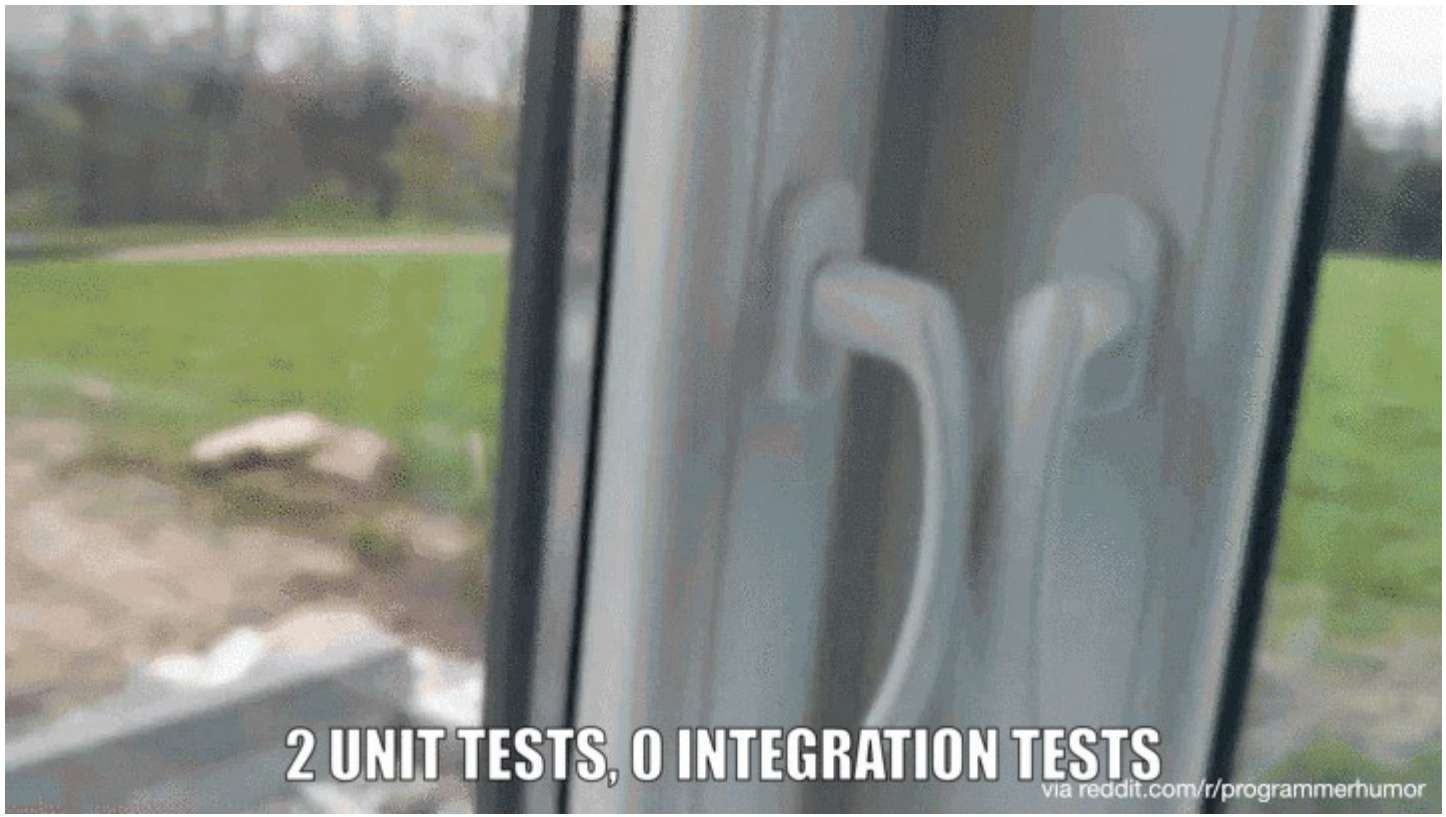
Exploratory testing (+)

- It’s agile, flexible and responsive
- Focus & time is spent on testing and not on writing test scripts
- The tester is in charge
- Finding more bugs
- Finding unexpected bugs

Exploratory testing (-)

- Require experienced testers
- Weak on:
  - Repeatability
  - Deterministic outcome (pass/fail)
  - Traceability
  - Documentation

Test automation


2 UNIT TESTS, 0 INTEGRATION TESTS
via reddit.com/r/programmerhumor

Test automation

- **Can be very useful in some contexts**
- **Very good at checking & confirming existing beliefs**
- **Can be very good for regression testing**
- **Often good at code (or unit) level checking**
- **But it's not a sliver bullet that can do all types of testing**
- **Normally weak on usability testing, GUI testing, and other types of testing where we need to explore**
- **Good automation almost everytime needs good exploration beforhand**

Suggested reading is this short paper (Test automation snake oil):
http://www.satisfice.com/articles/test_automation_snake_oil.pdf

Summary

Testing is really about...

Investigation and evaluation of a product in order to reveal information about how it might satisfy the customer, and why it might not satisfy the customer.

(*It's more than just checking stated requirements.*)

Agile Testing
@
Sectra

Development in

1. Linköping [HQ]
2. Örebro
3. Stockholm
4. Ipswich (UK)
5. Mansfield (UK)

SECTRA

Knowledge and passion

Secure Communication Systems

SECTRA

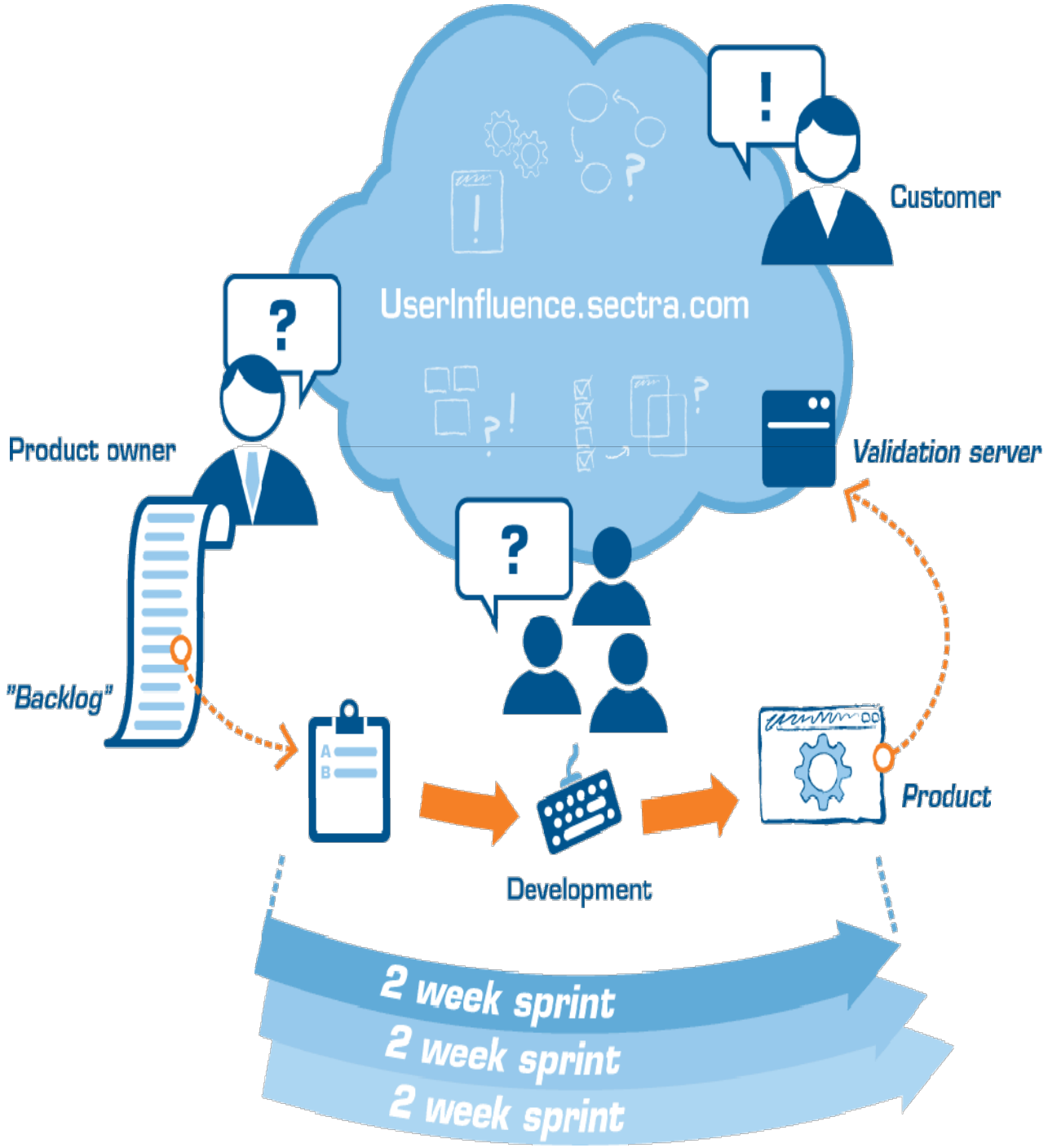
Knowledge and passion

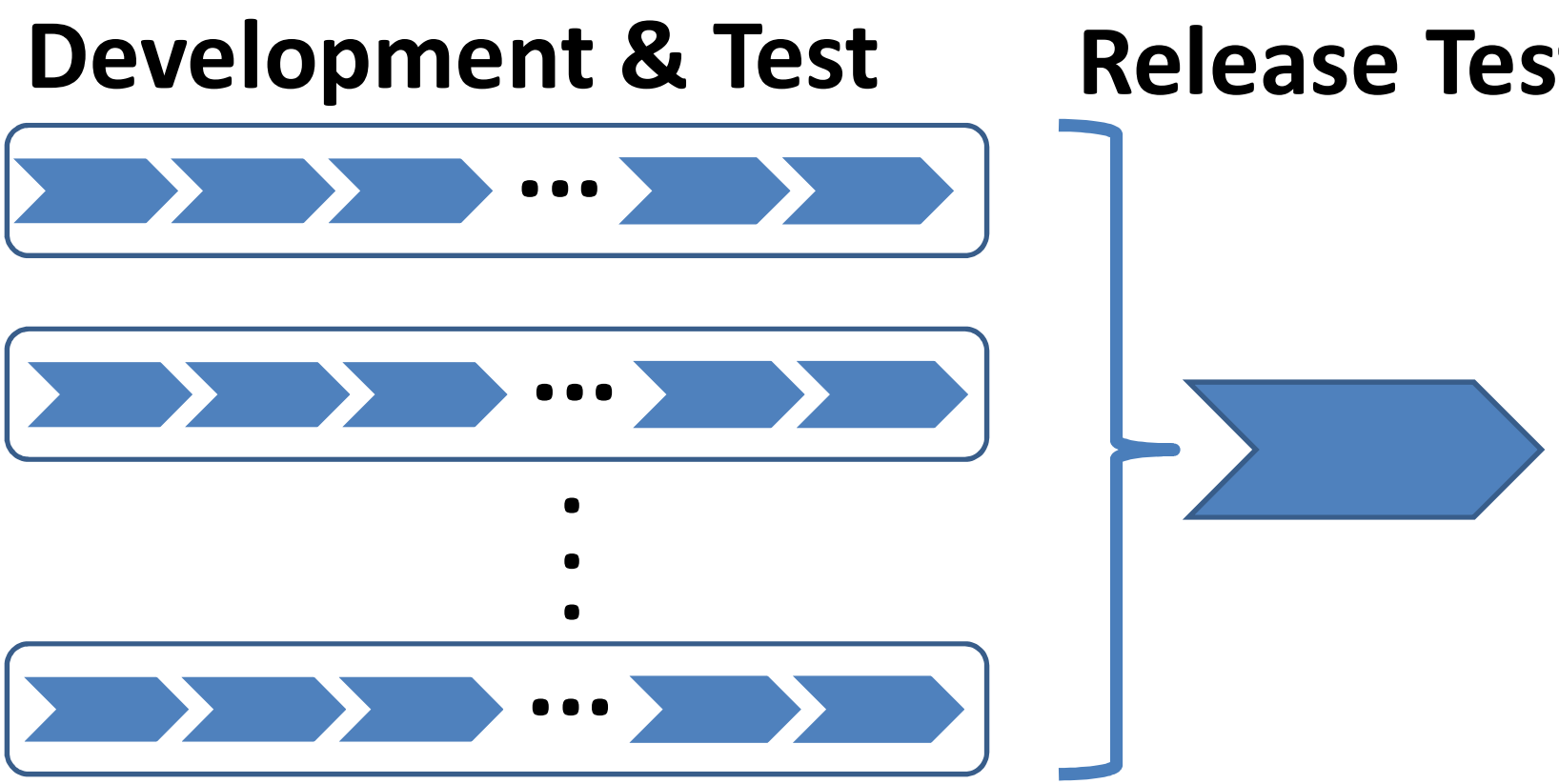
Medical Systems

Our mission is to increase effectiveness of healthcare, while maintaining or increasing quality in patient care.

15 Agile development teams

1-2 Testers + 3-4 Programmers per team


UserInfluence.sectra.com
Customer
Product owner
Validation server
"Backlog"
Development
Product
2 week sprint
2 week sprint
2 week sprint

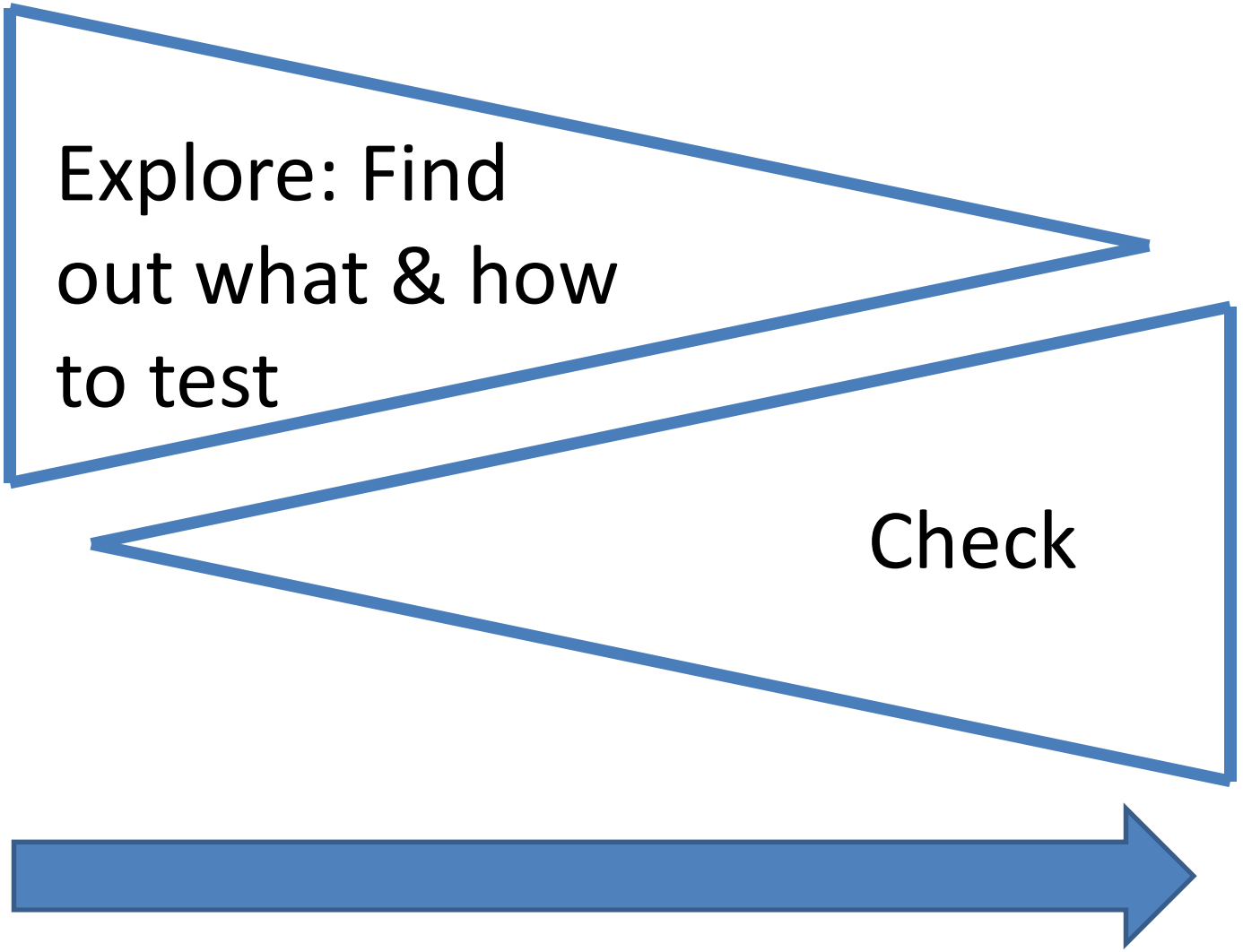


Testing
- During Sprints
- Between Sprints (Cross Team Testing)
- During Release Test

Testing in a sprint

- **TDD, Unit testing & Code reviews** to check the code (performed by developers)
- **Exploratory testing** to challenge the desing & to find bugs (perfomred by testers)
- **Focus is more on exploration than checking**
- **Outputs:**
  - **Updated (or new) regression tests (manual or automated)**
  - **Test cases for the Release Test phase**
  - **Bug reports (only unfixed bugs)**.

A sprint from a testing view



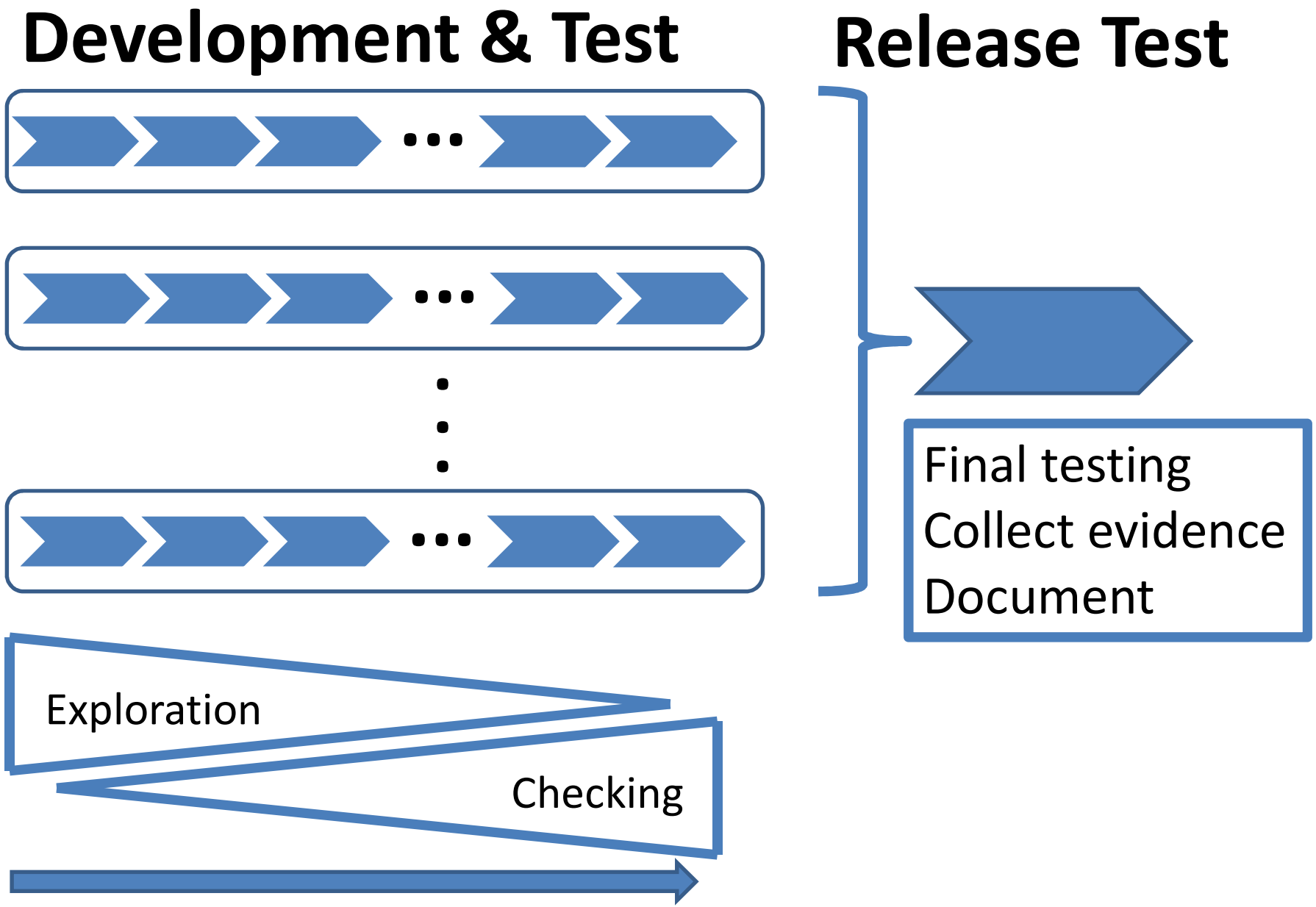
Cross Team Testing

Gather all testers to test each others test objects during every sprint.

We need to get fresh, unbiased, independent eyes on what is being developed.

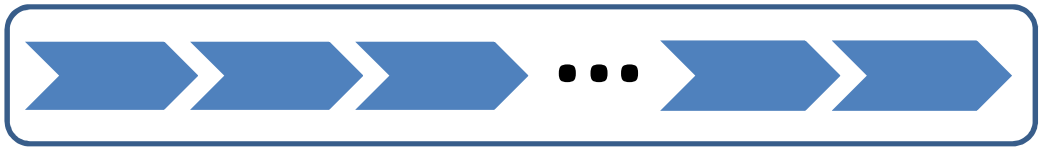
Release Testing

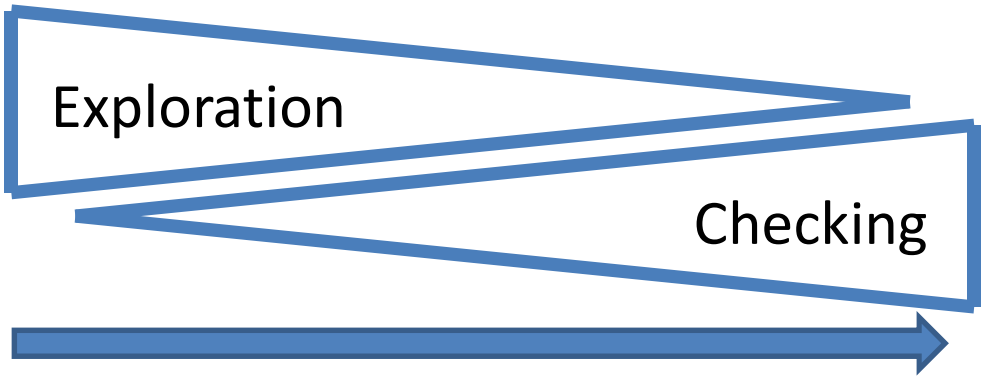
- Co-ordinated by a test project manager
- Mix of:
  - Re-test of new features,
  - workflow based tests,
  - regression tests
- Focus is more on checking than exploration
- Test environment is freezed & bigger


Development & Test
Release Test
Final testing
Collect evidence
Document
Exploration
Checking

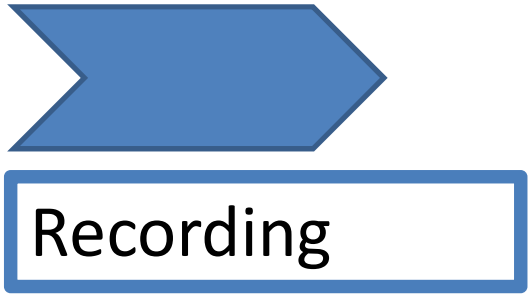
Orchestra analogy

Rehearsal





Consert

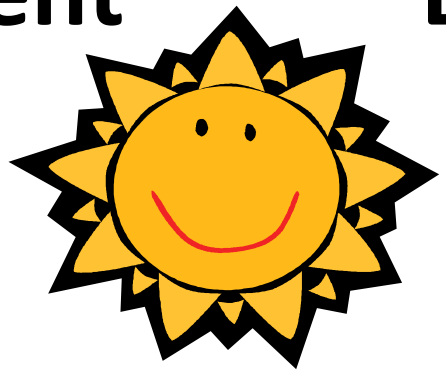


Benefits

of having testers in the development teams

Early involvement

Bug prevention



More Testing
(Less admin)

Better Quality

No walls between testers & programmers

Challenges

of having testers in the development teams

Walls between the teams

(i.e. between the testers)

Biased

(testing your own baby)

A lone tester

(Beeing the only tester in a team)

Summary

Testing is more than checking the stated requirements

(remember the software potato)

Video

Open Lecture by James Bach on Software Testing:

https://www.youtube.com/watch?v=ILkT_HV9DVU

This is a great lecture (1h 40min) by the most famous tester in the world (James Bach) about software testing. It's fun, educational and a must for anyone working with software development & testing.

Blogs & Articles

Testing Without a Map (Bolton) http://www.developsense.com/articles/2005-01-TestingWithoutAMap.pdf

Heuristic Test Strategy Model (Bach) http://www.satisfice.com/tools/satisfice-tsm-4p.pdf

Test Framing (Bolton, Bach) http://www.developsense.com/resources/TestFraming.pdf

Framing Test Framing (Bolton) http://www.developsense.com/blog/2011/05/framing-test-framing/

Better Software Magazine http://www.stickyminds.com/BetterSoftware/magazine.asp

RST Appendices (collection of articles, bibliografi, list of tools) (Bach) http://www.satisfice.com/rst-appendices.pdf

You Are Not Done Yet (Hunter) http://www.developsense.com/blog/2011/05/framing-test-framing/

The Value of Checklists (Kaner) http://www.kaner.com/pdfs/ValueOfChecklists.pdf

Touring Heuristic (Kelly) http://www.michaeldkelly.com/archives/50

When Do we Stop A Test? (Bolton) http://www.developsense.com/blog/2009/09/when-do-we-stop-test/

Checking vs. Testing (Bolton) http://www.developsense.com/blog/2009/08/testing-vs-checking/

Emotions And Oracles (Bolton) http://www.developsense.com/presentations/2007-10-STARWest-EmotionsAndTestOracles.pdf

Why Is Testing Taking So Long? (Bolton)

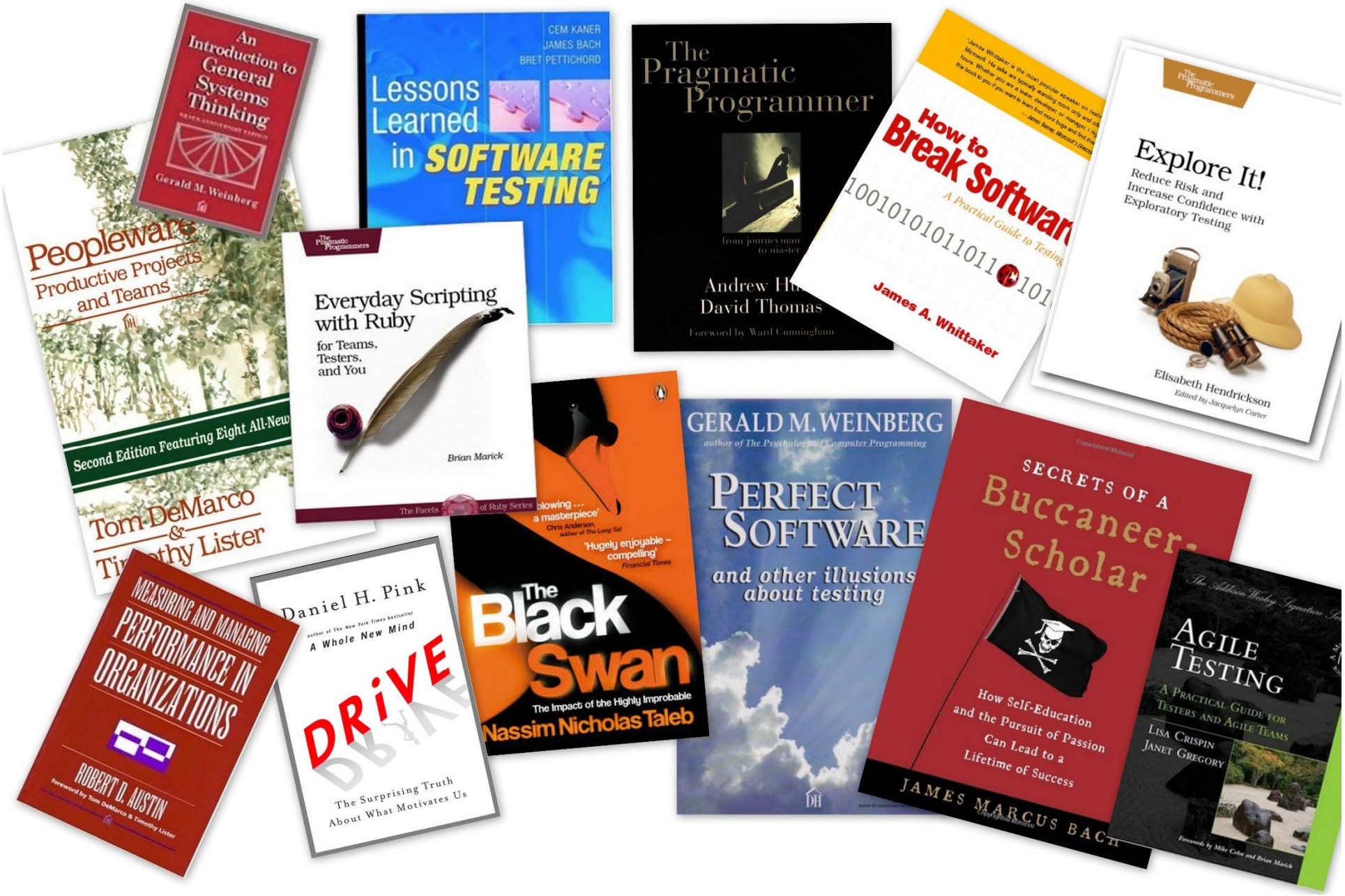
- http://www.developsense.com/blog/2009/11/why-is-testing-taking-so-long-part-1/
- http://www.developsense.com/blog/2009/11/what-does-testing-take-so-long-part-2/

The Case Against Test Cases (Bach) http://www.satisfice.com/presentations/againsttestcases.pdf

Test Heuristics Cheat Sheet (Data Type Attacks & Web Tests) (Hendrickson, Lyndsay, Emery) http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf

The Little Black Book on Test Design (Edgren) http://www.thetesteye.com/papers/TheLittleBlackBookOnTestDesign.pdf

Lightweight Characteristics Testing (Edgren) http://www.thetesteye.com/papers/LightweightCharacteristicsTesting.pdf


An Introduction to General Systems Thinking
Gerald M. Weinberg
Lessons Learned in SOFTWARE TESTING
CEM KANER
JAMES BACH
BRET PETTICHORD
The Pragmatic Programmer
Andrew Hunt
David Thomas
Foreword by Ward Cunningham
How to Break Software
A Practical Guide to Testing
James A. Whittaker
Explore It!
Reduce Risk and Increase Confidence with Exploratory Testing
Elisabeth Hendrickson
Edited by Jacquelyn Carter
Peopleware
Productive Projects and Teams
Second Edition Featuring Eight All-New
Tom DeMarco
&
Timothy Lister
Everyday Scripting with Ruby
for Teams, Testers, and You
Brian Marick
The Black Swan
The Impacts of the Highly Improbable
Nassim Nicholas Taleb
GERALD M. WEINBERG
author of The Psychology of Computer Programming
PERFECT SOFTWARE
and other illusions about testing
SECRETS OF A Buccaneer-Scholar
How Self-Education and the Pursuit of Passion Can Lead to a Lifetime of Success
JAMES MARCUS BACH
AGILE TESTING
A Practical Guide for Testers and Agile Teams
Lisa Crispin
Janet Gregory
MEASURING AND MANAGING PERFORMANCE IN ORGANIZATIONS
ROBERT D. AUSTIN
Daniel H. Pink
A Whole New Mind
DRIVE
The Surprising Truth About What Motivates Us

johan.atting@sectra.se
johan.jonasson@houseoftest.se
martin.gladh@frontit.se


SECTRA
Knowledge and passion